JOURNAL OF BASIC SCIENCE AND ENGINEERING

REACTIVE FAULT TOLERANCE IN CLOUD COMPUTING USING DEEP LEARNING AND HYBRID OPTIMIZATION TECHNIQUES

[1*]Malatesh Kamatar, [2]Dr. P Bindhu Madhavi

[1*]Department of Computer Science and Engineering, PDIT, Hosapete, India
[2]Department of CSE, T John Institute of Technology, Bangalore, India

DOI: https://doie.org/10.10399/JBSE.2025716014

Abstract - Cloud computing is becoming a sure technology for big applications, but its dynamic nature leads it to faults and inefficiency. This paper introduces a novel framework for reactive fault tolerance in cloud computing using deep learning and hybrid optimization to improve reliability and efficiency. The Hybrid Black-winged Kite with Lyrebird Optimization (HBKLO) algorithm is used to optimize scheduling of tasks and failure recovery by striking a balance between multi-objectives such as cost, makespan, and energy consumption. Dynamic resource usage is achieved by a Dynamic Cognitive Reinforcement Learning (DCRL)-based load balancer that reassigns the tasks across virtual machines in real-time. It applies the Modified Long Short-Term Memory (MLSTM) model for predicting faults in time using historical data and can allow proactive system management. In such a failure situation, the proposed framework also employs a reactive Failure Recovery mechanism with HBKLO algorithm that reassigns tasks dynamically on available VMs in such a way that there is minimal disruption and fast recovery, thus balancing resource utilization to enhance system reliability. The validated result proves the effectiveness of such an integrated approach for achieving scalable, cost-effective, and energy-efficient solutions in terms of managing dynamic workloads for modern cloud computing environments.

Keywords: *Reactive Fault Tolerance; Modified Long Short-Term Memory; Hybrid Black-winged Kite with Lyrebird Optimization; Dynamic Cognitive Reinforcement Learning and Failure Recovery.*

1. INTRODUCTION

Cloud computing is a valuable technology that is rapidly developing and providing consumers with access to resources at any given time. Fault tolerance dictates the execution of tasks and the availability of services [17]. Fault-tolerant scheduling strategies are required for managing performance variations, resource fluxes, and task execution failures in CC systems because they are prone to failures and fluctuations in resources [16]. The definition of a fault is a system's capacity to operate according to its design, even with defects or faults [10]. In cloud computing systems, techniques for fault tolerance must be used to ensure that there is high availability and high dependability [15]. With the advancement of internet-based applications, fault tolerance, performance, and throughput are some significant research issues in large-scale networks [22].

Large-scale applications are using cloud computing more and more, however because of the high burden on servers and components, failure rates can be substantial [7]. Implementing fault tolerance in cloud computing is difficult because of dependencies, complicated configurations, and changeable service infrastructures, despite the fact that it is essential for dependable services [5]. There are drawbacks to typical fault tolerance strategies in cloud computing, such as task replication decreasing performance under varying loads and checkpoint recovery lengthening waiting times [20]. The rising use of cloud computing creates problems with resource demand, which results in performance deterioration, load imbalance, outages, and excessive power consumption. High operating expenses result from the usage of many cloud and virtual

machine copies in current systems [19]. Additionally, it poses problems with sustainability, energy use, resource management, dependability, and security [4].

Threshold-Based Adaptive Fault Tolerance, which is an amalgam of proactive and reactive strategy, is defined. Its proactive approach uses VM migration as well as replication and retry for reactive [1]. With scientific workflow management and scheduling as a system that accommodates simultaneous execution, pipelined execution as well as task aggregatement of job qualities, cloud computing is recommended to be operated upon by a FD-SWMS [14]. Application of machine learning techniques to enhance cloud fault tolerance and database replication [13]. A new hybrid model was proposed, known as Hybrid Fault-tolerant Scheduling and Load Balancing Model (HFSLM), to optimize the makespan of tasks that are dynamically incoming and make efficient use of available virtual machines (VMs) in cloud systems [12]. The fault-tolerant approach utilizes strategic reallocation and deallocation of VM, which activates idle Virtual Machines (VM) dynamically through controlled QoS degradation [21].

The paper focuses on the highly advanced strategy for cloud computing systems' fault tolerance, employing techniques of deep learning and optimization so that the prediction regarding the faults can be better in task management. This study uses the Modified Long Short-Term Memory model so that it can accurately foresee failures based on the given historical data and the general behavior of the system. The technique integrates Multi-Objective Optimization to balance the loads in the system effectively and reduce resource wastage. A Load Balancer will automatically change the resource distribution in real time to reflect changing demands. The system also incorporates a Failure Recovery mechanism that automatically redistributes tasks upon failure, thereby reducing the system's downtime and making it more reliable. By minimizing service outages and making the most use of available resources, it significantly boosts cloud environments' resilience and performance.

The major contributions of the paper are as follows:

Developed the Hybrid Black-winged Kite with Lyrebird Optimization (HBKLO) algorithm to optimize task scheduling and failure recovery, balancing multiple objectives such as cost, makespan, and energy consumption.

Proposed a Dynamic Cognitive Reinforcement Learning (DCRL)-based load balancer that dynamically reallocates tasks among virtual machines in real-time, ensuring optimal resource utilization under varying workloads.

Introduced the Modified Long Short-Term Memory (M-LSTM) model for accurate and timely fault prediction in cloud computing using historical data, enabling proactive system management.

The following sections are organized as follows: Section 2 explores relevant research and literature reviews, Section 3 introduces the proposed framework, Section 4 provides detailed analysis of the observed results and discussions, and Section 5 offers the final assessment of this study.

2. LITERATURE SURVEY

Some of the recent research works related to the Reactive Fault Tolerance in Cloud Computing were reviewed in this section

Kalaskar and Thangam (2023) [8] applied ML models for cloud performance prediction and fault tolerance improvement. Hence, they discussed some of the essential issues related to cloud computing. For designing prediction models, we have implemented machine learning methods sequentially, which include

gradient boosting, decision trees, and linear regression (LR). For cloud activity prediction, these models performed better than the baseline methods. C5.0 and XGBoost showed high accuracy, precision, and reliability. Feature importance analysis is conducted in order to find out the top ten most important features affecting cloud system performance. This work significantly contributes toward optimizing clouds and making them more reliable; proactive monitoring of the systems is possible, leading to the early detection of performance degradation and fault tolerance.

Mehta et al. (2021) [11] introduced the diagnosis of bearing faults using IRT. A 2D discrete wavelet transform (2D-DWT) is used to deconstruct the thermal picture. Subsequently, PCA reduces the dimensionality of the extracted feature and gives the most relevant features. Additionally, for fault classification and performance evaluation, classifiers such as linear discriminant analysis (LDA), support vector machines (SVM), and k-nearest neighbor (KNN) were also taken into consideration. As per the results, the SVM outperformed the LDA and KNN.

Devi and Paulraj (2021) [3] have reduced system crashes in real-time systems employing multi-level fault tolerance scheduling technique. The method applies teaching-learning based optimization scheduling mechanism in high availability and reliable K-nearest neighbor decision rule for reliability. Response time, performance improvement rate, failure rate, makespan time, and also rejection ratio makes a benchmark of the evaluation method at a Cloudsim platform. The findings reveal that data in the multi-level structure is very much available as well as dependable in the cloud environment.

Uppal et al. (2022) [24] presented a prediction model that checks real-time data from sensor nodes in a clinical setting with the help of a machine learning algorithm. The Internet of Things (IoT)-based smart hospital environment is developed, where numerous appliances are controlled and monitored with the help of the Internet through a variety of sensors, including ultrasonic, flame, temperature, humidity, and current sensors. Three crucial features of sensor data created by the Internet of Things are its vast volume, structure, and real-time nature. To guarantee the fidelity, accuracy, dependability, and integrity of devices with IoT capabilities, the primary goal of this research is to anticipate early defects in an IoT context. Using Decision Tree (DT), Random Forest (RF), Gaussian Naive Bayes (GNB), and K-nearest neighbor (KNN) methods, the proposed model for fault prediction was evaluated, nonetheless, RF demonstrated the highest accuracy.

Zhao et al. (2021) [25] presented UDFP (Unlabeled Data based online Failure Prediction), an online failure prediction framework technique that achieves prediction modeling through a clustering analysis method that combines modularity and KNN. In safety-conscious distributed cloud data centers, the method can save data administration expenses, enhance predictive accuracy, and lessen supervised learning issues for failure prediction. According to experimental results, UDFP enhances fault-tolerant capabilities and robustness while avoiding the workload associated with human tagging, improving prediction accuracy, and lowering data management expenses.

Krishna and Mangalampalli (2023) [23] gathered the priorities of jobs provided to the scheduler for VMSs based on the cost per unit of electricity. Regarding this priority given to the scheduler, this scheduler is abstracted using DQN: a deep technique of reinforcement learning for decision making and best possible generation of schedules for the virtual machine. Many researches were performed regarding CloudSim.

Sathiyamoorthi et al. (2021) [18] tried to offer fault-free task scheduling by proposing a flexible and efficient fault-tolerant scheduling technique. In order to achieve the best quality of service, the presented

Vol. 22, No. 1, (2025)
ISSN: 1005-0930

approach considers the most significant factors: the failure rate and the resources' current workload. The method recommended is verified using commonly used metrics such as the usage of resources, average throughput, success rate, makespan, and execution duration by using the CloudSim toolkit. As revealed by the empirical results, the proposed strategy outperforms the benchmark strategies in fault tolerance and load balancing.

Kruekaew and Kimpan (2022) [9] proposed an autonomous cloud-based system for scheduling tasks that optimizes multi-objective task scheduling by combining the Artificial Bee Colony Algorithm (ABC) with a Q-learning algorithm—the reinforcement learning methodology speeds up the ABC algorithm—is known as the MOABCQ method. With an intention to overcome the limitation of concurrent considerations, the proposed solution looks to improve VM throughput and also optimize scheduling and resource utilization with the help of providing load balancing amongst the VMs according to resource usage, cost, and makespan. For the purpose of performance study, CloudSim has been used to compare the performance of the proposed approach against the existing load balancing and scheduling algorithms.

Jena et al. (2022) [6] presented a novel dynamic load balancing technique for virtual machines which incorporates altered particle swarm optimization with an improved Q-learning technique called QMPSO. The hybridization process in order to maximize throughput and maintain the task priorities through optimization of waiting time and also balance load among virtual machines (VMs) modifies MPSO velocity through gbest and pbest. By comparing the results obtained from QMPSO using the additional scheduling and load balancing techniques, it confirms the robustness of the algorithm.

Alghamdi (2022) [2] presented PSO's binary version for balancing and scheduling the workload of cloud computing systems. Significant variation in completion times over heterogeneous virtual machines is determined by the objective function. An approach to particle placement update is devised that outperforms current heuristic and metaheuristic algorithms in load balancing and job scheduling. Because artificial neural networks (ANN) are faster and more accurate at predicting targets than multilayer perceptron networks, they have been used to attain this result. The comparison table for the literature review papers are shown in Table 1.

Table 1: Comparison of the existing papers

Author's Name	Aim	Methods	Advantages	Disadvantages
Kalaskar and Thangam (2023) [8]	To enhance cloud performance forecasting and fault tolerance in cloud computing systems by leveraging machine learning models.	DT, LR, XGBoost	Handling Imbalanced Data, Real-World Applicability	Dependence on Specific Features
Mehta et al., (2021) [11]	To develop a robust and efficient methodology for bearing fault diagnosis using IRT.	SVM, KNN	Dimensionality Reduction, Effective Feature Extraction	Computational Complexity, Real-Time Limitations
Devi, and Paulraj (2021) [3]	To create a cloud computing multi-level fault-tolerance scheduling system	RDK-NN, TLBO	Scalability and Applicability	Generalization Issues
Uppal et al. (2022) [24]	To develop a fault prediction model for monitoring real-time IoT sensor data in a clinical environment, ensuring enhanced IoT-enabled devices' accuracy, dependability, fidelity, and integrity.	RF, KNN, DT, GNB	Cost and Time Savings, Real-Time Monitoring	Scalability Challenges, Limited Dataset Size

Zhao et al., (2021) [25]	To improve fault tolerance and enhance the robustness of systems by predicting failures proactively	KNN	Real-Time Prediction, High-Efficiency Clustering	Limited to Unlabeled Data
Krishna, and Mangalampalli (2023) [23]	To improve cloud computing's operational efficiency by ensuring optimal task scheduling that maximizes resource utilization while minimizing energy consumption and downtime.	DQN	Energy Efficiency, Dynamic Decision-Making	Complexity of DQN Model, Limited Task Forecasting
Sathiyamoorthi, et al., (2021) [18]	To suggest a fault-tolerant, adaptive scheduling strategy for cloud computing settings.	QoS	Reliability and Throughput	Lack of Dynamic Adaptability
Kruekaew, and Kimpan, (2022) [9]	To tackle the difficulties associated with job scheduling and workload balancing in cloud computing systems	Q-learning, ABC	Versatile and Adaptable	Need for Real-World Validation, Potential for Complexity
Jena et al., (2022) [6]	To suggest a better approach to load balancing in cloud computing settings	MPSO, Q-learning	Reduced Waiting Time,	Complexity of Hybridization
Alghamdi, (2022) [2]	to create a cloud computing load balancing and task scheduling technique that works	ANN	Low-Cost and Low-Complexity Solution	Limited to Non-Dependent Tasks

a. Research gap

The use of a reduced-scale version of Google trace data limits generalizability to actual cloud environments. The dynamic imbalances may not be well captured by the label imbalance approaches. Exploration of potentially superior models is limited to only five algorithms in this study. Further, more accurate or domain-specific measurements might increase the accuracy even as important aspects like "cpu_usage_distribution" are considered. The absence of external validation and testing on bigger datasets weakens the robustness of the findings. SVM is highly sensitive to tuning, and the fault classification system may miss significant characteristics due to its reliance on thermal imaging and PCA's dimensionality reduction. The multi-level fault-aware scheduling technique also requires more research for scalableness in hybrid context since it has issues concerning the accuracy of fault detection, performance degradation under higher loads, and lack of attention to energy, security and cost optimization.

3. PROPOSED METHODOLOGY

The proposed methodology incorporates a reactive fault tolerance framework into cloud computing using deep learning and hybrid optimization techniques. Proactive system management is achieved with the help of a model that predicts faults using historical data, which is the Modified Long Short-Term Memory (M-LSTM) model. Task scheduling along with failure recovery is optimized using the Hybrid Black-winged Kite

with Lyrebird Optimization (HBKLO) algorithm, which balances the multi-objectives like cost, makespan, and energy consumption. This dynamic load balancer will make use of a DCRL-based framework that will learn and adapt its task distribution in real-time by using attention mechanisms, episodic memory, and meta-cognitive adaptation across virtual machines. The HBKLO algorithm is very fast, allowing the rapid reassignment of tasks during failures while minimizing disruption to maintain the Quality of Service. The block diagram is shown in Figure 1. The approach ensures scalability, economy of operations, and also ensures energy efficiency, providing for greater reliability and efficiency for dynamic cloud computing environments.

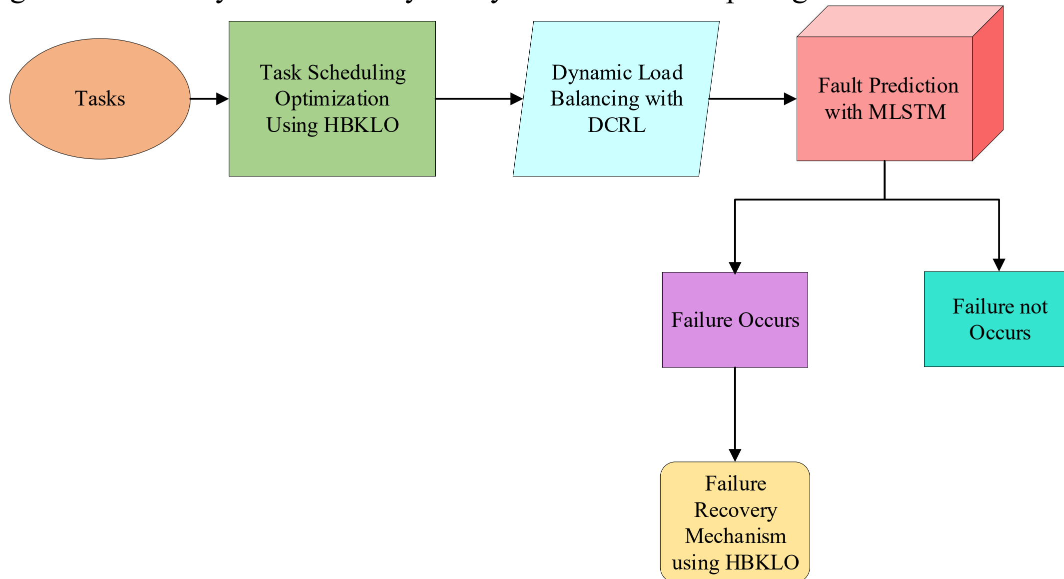


Figure 1: Flow diagram of the proposed fault tolerance mechanism

a. Multi-Objective based Scheduling of tasks

Multi-objective-based task scheduling for cloud computing aims at optimal scheduling of tasks in cloud computing to balance three key objectives-cost, makespan, and energy consumption. The scheduling is made by using a multi-objective fitness function, such that the total execution cost, including CPU and memory usage is minimized, makespan is reduced, and also energy consumption is minimized. The HBKLO algorithm is used for dynamic task assignment to virtual machines, taking into consideration the available resources, execution time, and energy efficiency. The scheduling mechanism ensures efficient resource utilization, cost-effectiveness, and reduced operational time by adjusting the priorities of these objectives, thus improving the system's performance and scalability in dynamic cloud environments.

i.Design of multi-objective

The fitness function quantifies the degree to which a particular solution fits the problem, as it is intended to assess the solution as well. The multi-objective function for fitness evaluation is introduced in this study. There is only one parameter that the single objective function is concerned with. The system's overall performance is impacted by this. In this paper, the multi-objective function is so created. Three factors—energy, makespan, and overall execution cost—are used in the design of the suggested MOF. The parameter indicates that there is a minimization problem with the suggested multi-objective function. In Eq. (1), the suggested fitness function is provided.

$$Fitness = min[\alpha_1 \times \cos t + \alpha_2 \times makespan + \alpha_3 \times Energy] \quad (1)$$

Cost is the first argument in the objective function. The amount of money used for the scheduling procedure is the basis for measuring total execution cost (TC). When a user wishes to use the resources in a virtual machine

(VM), they must pay for the task. The payment includes the entire cost of the CPU and memory if the job W_i is scheduled as VM_j. Eq. (2) is utilized to determine the CPU cost.

$$C_{cost}(j) = C_{base} \cdot T_{ij} \cdot C_j \quad (2)$$

Where T_{ij} is the task W_i's execution time on VM_j, C_j and C_{base} is the total amount of CPUs and C_{base} is the CPU base cost. Eq. (3) is used to compute the memory cost.

$$M_{cost}(j) = M_{base} \cdot T_{ij} \cdot M_j \quad (3)$$

The total amount of memory is M_j. Memory base cost, abbreviated M_{base}. Eq. (4) evaluates the total cost of completing all jobs.

$$TC = \sum_{i=1}^{N} C_{cost}(j) \cdot X_{ij} + \sum_{i=1}^{N} M_{cost}(j) \cdot X_{ij} \quad (4)$$

The assignment matrix is located in X_{ij}. Makespan is the second parameter in the fitness function. Makespan is the most important task scheduling parameter. The goal of scheduling is to schedule the full task with the resources available in the shortest amount of time possible. The makespan is computed by means of Eq. (5).

$$Makespan = CT_{Max}[i,j]\ i \in T, i = 1, \ldots, n\ j \in VM, i = 1, \ldots., k \quad (5)$$

Where CT_{Max} is calculated using Eq. (6)

$$CT_{Max} = \sum_{i=1}^{N} \sum_{j=1}^{k} (T_{ij} \cdot X_{ij}) \quad (6)$$

Where CT_{Max} is the maximum completion, n represents the task count, k is the visual machine count. Energy is the objective function's third parameter. To arrange the full activity using the resources at hand, an effective scheduling system expends minimal energy. Eq. (7) provides the total energy used during scheduling.

$$E_{total} = E_{busy} + E_{idle} \quad (7)$$

When every resource is in use, energy consumption is determined using Eq. (8)

$$E_{busy} = \sum_{i=1}^{n} K \times V_{ip_j,S}^2 \times f_{r_j,S} \times T_{i,j} = \sum_{i=1}^{n} P_{dynamic,i} \times T_{i,j} \quad (8)$$

Where $T_{i,j}$ is the time to use resource r_j to complete the task. Depending on the voltage $V_{ip_j,S}^2$, the task is scheduled to be completed in r_j. The frequency of resource r_j having voltage level s. Eq. (9) is used to compute energy usage when all resources are idle.

$$E_{idle} = \sum_{j=1}^{p} K \times V_{i_{lowest}}^2 \times f_{j_{lowest}} \times T_{j_{idle}} \quad (9)$$

In this case, V is the lowest voltage of the resources, and f is the lowest frequency. $T_{j_{idle}}$ is the resource j_{idle} time.

ii.Scheduling of tasks using HBKLO algorithm

The BKA is an advanced metaheuristic algorithm. The black-winged kite's survival strategy serves as an example. They have extraordinary hovering abilities and are excellent hunters. They consume birds, small mammals, insects, and reptiles. An algorithmic model was developed based on the hunting abilities and flying patterns of the black-winged kite. It is used for scheduling the tasks in available VMs.

1. Initialization phase

The initialization of the population in BKA begins with the creation of a collection of random solutions. The position of each Black-winged kite (BK) may be represented using the following matrix.

$$P = \begin{bmatrix} P_{1,1} & \ldots & P_{1,d} & \ldots & P_{1,d} \\ P_{2,1} & \ldots & P_{2,d} & \ldots & P_{2,d} \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ P_{S,1} & \ldots & P_{S,1} & \ldots & P_{S,d} \end{bmatrix} \quad (10)$$

P_{ij} represents the jth dimension of the ith Black-winged kite, d represents the dimension of the issue, and S represents the number of possible solutions. Every Black-winged kite's location is consistently assigned in this text.

$$A_i = P_L + r(P_U - P_L) \tag{11}$$

Where i is an integer between 1 and S, P_L and P_U are the lower and upper limits of ith Black-winged kites in the jth dimension, respectively, and r is a randomly chosen number between [0, 1]. By selecting the individual with the best fitness value as the leader A_L in the initial population, BKA determines the optimal location for Blackwinged kites. This is an illustration of how the lowest value may be used to mathematically represent the original leader A_L.

$$F_b = \min(F(A_i)) \tag{12}$$

$$A_L = A\left(find\left(F_B == F(A_i)\right)\right) \tag{13}$$

2. *Attacking behavior*

The black-winged kite monitors its target, adjusting its wings and tail to match the wind speed before diving to attack. The original BKA algorithm struggles with balancing diversity and convergence, limiting its effectiveness. To address this, a dynamic nonlinear convergence factor is introduced, which adjusts throughout the iterative process. Weights are gradually reduced in subsequent iterations, allowing the algorithm to focus on explored areas, enhancing local search capabilities and accelerating convergence.

$$a_{i,j}^{T+1} = \begin{cases} a_{i,j}^T + N.\left(1 + sin(r)\right) \times a_{i,j}^T, & k < r \\ a_{i,j}^T + N \times (2r - 1) \times a_{i,j}^T, & else \end{cases} \tag{14}$$

$$N = 0.05 \times e^{-2\times\left(\frac{T}{itr}\right)^2} \tag{15}$$

Where $a_{i,j}^T$ and $a_{i,j}^{T+1}$, respectively, indicate the locations of the ith black-winged kite in the jth dimension in the Tth and $(T+1)^{th}$ iterations. itr is the total number of iterations, T is the number of iterations that are now occurring, r is a random number between 0 and 1, and k is a constant value of 0.9.

3. *Migration Behaviour*

During the exploitation stage, BKA simulates the complex migration behavior of black-winged kites. The Leader strategy is used, where if the current population's fitness is lower than that of a random population, it loses its leadership and becomes part of the migratory population. Conversely, if the current population has a higher fitness, migration is directed in that direction. This dynamic leadership selection enhances the migration's success. The black-winged kite migration behavior can be mathematically represented as:

$$a_{i,j}^{T+1} = \begin{cases} a_{i,j}^T + \mathbb{C}(0,1) \times \left(a_{i,j}^T - q_j^T\right), & \mathcal{F}_i < \mathcal{F}_{ri} \\ a_{i,j}^T + \mathbb{C}(0,1) \times \left(q_j^T - m \times a_{i,j}^T\right), & else \end{cases} \tag{16}$$

$$m = 2 \times \sin(r + \pi/2) \tag{17}$$

Whereas q_j^T indicates the score of the black kite leader in the jth dimension in the T^{th}iteration thus far. $C(0,1)$ denotes the Cauchy mutation, which is defined as follows: F_i is the fitness value of any individual in the Tth iteration, F_{ri} is the fitness value of any black-winged kite in the Tth iteration:

$$\mathcal{F}(a, \delta, \mu) = \frac{1}{\pi}\frac{\delta}{\delta^2 + (x-\mu)^2} \qquad , -\infty < a < \infty \tag{18}$$

The Cauchy mutation expression when $\delta = 1$ and $\mu = 0$ is:

$$\mathcal{F}(a, \delta, \mu) = \frac{1}{\pi}\frac{1}{a^2 + 1} \qquad ,-\infty < a < \infty \tag{19}$$

4. *Hiding Strategy*

The placements of population members within the search space are adjusted during this phase of HBKLO to align with the lyrebird's evacuation strategy to a nearby safe haven. Using this strategy, the BKA is gradually changes where it is as it explores its surroundings cautiously and walks slowly in search of a good hiding place. This demonstrates how local search operations employ LYBA. Each member's new position is determined by HBKLO simulating the lyrebird's migration toward a nearby suitable hiding place using Eq. (20). The goal function value carried by the linked member is replaced with the new location if it increases if Eq. (20) holds true.

$$a_{i,j}^{T+1} = a_{i,j}^{T} + \left(1 - 2r_{i,j}\right).\frac{U_j - L_j}{T} \quad (20)$$

The iteration number in this case is T, the new position of the ith lyrebird is chosen using the suggested HBKLO's concealing method, $a_{i,j}^{P2}$ represents the jth dimension, F_i^{P2} represents the value of the objective function, and random numbers $r_{i,j}$ from the range [0, 1].

b. DCRL for load balancing

DQN for load balancing utilizes reinforcement learning for immediate task reallocation among various virtual machines to achieve optimal usage. Here, the state may be defined by such quantities as CPU and memory utilizations, while actions can be taken like the redistribution of tasks from the first virtual machine to the second and others, balancing the load among multiple virtual machines. DQN learns to maximize these long-term rewards, like more performance and efficiency. The core of the method is the Q-value function, which estimates the expected future rewards for each action, guiding the agent in making optimal decisions. The DQN ensures that tasks are distributed efficiently by continuously updating the Q-values based on real-time feedback, reducing latency and maximizing throughput, thus enhancing overall system performance in dynamic cloud environments. Q-learning is at the core of DQN. Determining the Q function is the fundamental idea behind Q-learning.

$$Q: \emptyset(S_t) \rightarrow Q(\emptyset(S_t), a_t; \theta) \quad (21)$$

It may be used to calculate the predicted accumulated rewards for acting at a given input $\emptyset(S_t)$, where $\emptyset(S_t)$ is the state reformulation and θ is the action-value function that maps the input to output decisions. After obtaining the Q function, a policy $\pi(s)$ may be constructed that maximizes the rewards by

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(S, a) \quad (22)$$

Matrix-based equations may be used to calculate Q for applications with a simple state, such as navigation in a basic grid maze. Modelling the Q-learning process is a difficult problem for this research.

i.Episodic Memory Integration

To improve long-term decision-making, we add an episodic memory module to the learning process. This module enables the agent to store and recall prior experiences. To avoid catastrophic forgetting and promote positive transfer, the episodic memory module is utilized for local adaptation and sparse experience replay. A key-value store is used to model this memory:

$$\mathcal{M} = \{(key_i, value_i)\} \quad (23)$$

Where key_i encodes the representation of the state S_t, $value_i$ stores Q values or action probabilities. The most pertinent memory is recovered during decision-making by calculating a similarity score between the stored keys and the current state.

$$Similarity: \kappa(S_t, key_i) = exp(-\|S_t - key_i\|^2) \quad (24)$$

The agent can concentrate on important prior experiences with the aid of this similarity metric. To enhance decision-making, recovered memory is added to the Q-value:

$$Q_{aug}(S_t, a_t) = Q(S_t, a_t) + \lambda \sum_i \kappa(S_t, key_i) \cdot value_i \quad (25)$$

Where the weighting factor λ regulates the memory's influence. The learning process is accelerated by the episodic memory, which aids in improved generalization by bringing up important memories when comparable conditions arise.

ii.Attention-Enhanced State Representation

By concentrating on the most pertinent aspects of the input state, employ an attention mechanism to improve the state representation. This is particularly helpful in settings when the incoming data contains unnecessary information or has a large dimensionality. A weight α_i is allocated to each feature f_i in the state and is calculated as follows:

$$\alpha_i = \frac{\exp(w_i h_t)}{\sum_j exp(w_i h_t)} \tag{26}$$

Where h_t is the hidden state from the input layer, and w_i are learnable weights associated with the features. Next, the attention-enhanced state is computed as follows:

$$S_t^{att} = \sum_i \alpha_i f_i \tag{27}$$

This enhances decision-making and lessens the impact of distracting or loud input by allowing the agent to concentrate on the most crucial elements of the surroundings. The attention-enhanced state representation is then used to update the Q-value function:

$$Q(S_t^{att}, a_i) = r_t + \gamma \max_{a'} Q\,(S_{t+1}^{att}, a') \tag{28}$$

As a result, the agent is more equipped to learn from pertinent state aspects.

iii.Meta-Cognitive Adaptation

To make the agent more adaptive in dynamic environments, a meta-cognitive module is introduced. This module dynamically adjusts the exploration rate (ϵ) and other hyperparameters based on the agent's performance. The meta-policy $\pi_{meta}(\theta_t)$ maximises the anticipated reward in order to optimize the hyperparameters.

$$\pi_{meta}(\theta_t) = arg \max_{\theta} \mathrm{E}_{\pi\theta}\,[R_t] \tag{29}$$

Where θ_t stands for the learning parameters, including reward scaling factors and exploration rate. The meta-policy's update rule is:

$$\theta_{t+1} = \theta_t + \eta \nabla_{\theta_t} \mathrm{E}_{\pi\theta}[R_t] \tag{30}$$

Where the learning rate for modifying the meta-policy parameters is represented by $\eta\nabla_{\theta_t}$. As a result, the agent can improve its ability to adapt to different settings by gradually modifying its learning method.

iv.Hierarchical Memory Replay

We employ hierarchical memory replay, which blends short-term and long-term memory, to improve learning even more. While long-term memory retains important prior experiences that might be helpful in making judgements in the future, short-term memory retains current events.

- Short-term Memory (D_{short}): Stores experiences from recent time steps (s_t, a_t, r_t, s_{t+1}).
- Episodic Memory (M): Stores critical long-term experiences.

The total loss function used for training combines both memories:

$$\mathcal{L}_{total} = \mathrm{E}_{(s,a,r,s')\sim D}\left[\left(Q(s_t, a_t; \theta) - \left(r_t + \gamma \max_{a'} Q\,(s_{t+1}, a'; \theta^-)\right)\right)^2\right] \tag{31}$$

Training is concentrated on more informative transitions by emphasizing encounters with larger temporal-difference (TD) mistakes through the use of a prioritized sampling strategy:

Vol. 22, No. 1, (2025)
ISSN: 1005-0930

$$P(i) = \frac{|\delta_i|^\beta}{\sum_k |\delta_k|^\beta} \tag{32}$$

Where β is a hyperparameter that regulates the degree of prioritisation, and $|\delta_i|^\beta$ is the TD error for experience i.

v.Self-Supervised Auxiliary Tasks

Self-supervised auxiliary activities are introduced to improve the agent's generalization and feature learning. The purpose of these exercises is to assist the agent develop more resilient representations and enhance its comprehension of the surroundings.

1. *State Prediction*

The agent predicts the next state $S_{t+1} - \hat{S}_{t+1}$ based on its current state:

$$\mathcal{L}_{state} = \left\| S_{t+1} - \hat{S}_{t+1} \right\|^2 \tag{33}$$

2. *Reward Estimation*

The agent predicts the immediate reward $r_t - \hat{r}_t$

$$\mathcal{L}_{reward} = \| r_t - \hat{r}_t \|^2 \tag{34}$$

The final total loss is a combination of the primary Q-learning loss and the auxiliary task losses:

$$\mathcal{L}_{final} = \mathcal{L}_Q + \lambda_1 \mathcal{L}_{state} + \lambda_2 \mathcal{L}_{reward} \tag{35}$$

Where λ_1 and λ_2 are weights that control the importance of the auxiliary tasks.

c. Fault tolerance in cloud computing

Cloud computing relies on fault tolerance mechanisms in ensuring high system reliability and availability despite hardware or software failures. In cloud environment, there are several possible kinds of failures, that could be virtual machine crashing, network failure, and failure of storage. A solution for these issues is developing the mechanisms of fault tolerance: this means detecting and recovering quickly from faults for continued service. In this proposed framework, fault tolerance would be achieved through proactive prediction of faults using a model called M-LSTM model and reactive failure recovery mechanisms by utilizing the HBKLO algorithm.

Let P be physical nodes in the cloud system as $PS = \{PS_1, PS_2, , PS_P\}$. It would signify the VMs which already exist in PS_i with $PS_i = \{VM_{1(i)}, VM_{2(i)}, \ldots, VM_{m(i)}\}$ having m_i be the count of VMs existing in that i-th physical node. $VM = \{VM_1, VM_2, , VM_M\}$ be set of M virtual machines. Jobs would be presented by the equations, $Job = \{job_1, job_2, ., job_J\}$. And also, J refers to the count of jobs. Then, some tasks which occur in i-th job have been incorporated by, $job_i = \{task_{1(i)}, task_{2(i)}, .., task_{j(i)}\}$. In addition, here $j(i)$ denotes the number of tasks in that particular i-th job. Overall, in this study, there is a set consisting of N task as $Task = \{T_1, T_2, ., T_N\}$ that should be assigned to the intended resource. The response time is an interval between sending a request and the first response generated by VMs. If T_i^{sent} i be the sent time and $T_i^{response}$ be the time of first response, then rt_i is the time of responding to the T_i task. The rt_i value as described in Eq. (36).

$$rt_i = T_i^{response} - T_i^{sent} \tag{36}$$

Additionally, rt is the average total response time, which is determined using Eq. (37) and Eq. (38).

$$\overline{rt} = \frac{1}{P} \sum_{j=1}^{P} rt(PS_j) \tag{37}$$

$$rt(PS_j) = \sum_{i=1}^{N_j} rt_i \tag{38}$$

The jobs allocated to the physical node PS_j are denoted by N_j, whereas P is the number of physical nodes. Let TS represent the entire time period. Since the physical node load is comparatively constant over time, the physical node load PS_i is determined for the k time period using $V(k, PS_i)$ and the number of tasks in its

Vol. 22, No. 1, (2025)
ISSN: 1005-0930

queue. $V(TS, PS_i)$ is specifically the PS_i physical node's load across the entire duration. Eq. (40) displays the physical nodes' average load.

$$\bar{V}(TS) = \frac{1}{P}\sum_{i=1}^{P} V(TS, PS_i) \quad (39)$$

To characterize the dispersion (intensity) of the load on various physical nodes, Eq. (40) is used to define the variance of the load on physical nodes.

$$\sigma(TS) = \sqrt{\frac{1}{P-1}\sum_{i=1}^{P}\left(V(TS, PS_i) - \bar{V}(TS)\right)^2} \quad (40)$$

i.MLSTM for fault detection

The MLSTM model is made up of a number of LSTM units that are grouped together in the LSTM layer. Three multiplicative units are present in LSTM models. First, the current information state is memorized using the input gate. Second, the results are shown using the output gate. Third, some lost historical content is selected using the forget gate. The sigmoid function and the dot product operation are examples of multiplicative units. The sigmoid function has a range of zero to one, and the dot product operation establishes the amount of data that must be transmitted. When the value of a dot product operation is 1, information is transmitted; when the value is 0, no information is communicated. The initial step in the MLSTM process is deciding which cell state data to eliminate. That decision is determined by the sigmoid layer, also referred to as the "forget gate layer." After examining h_{t-1} and x_t, the cell state C_{t-1} yields a number between 0 and 1 for each value. where 0 denotes the entire elimination of something and 1 denotes its full retention.

$$f_t = \sigma(W_f.\,[h_{t-1}, x_t] + b_f) \quad (41)$$

To change the state of the data stored $\tilde{C}_t$, the input gate layer determines which parameters should be upgraded and is coupled to the tanh layer, which creates a vector of updated potential solutions.

$$i_t = \sigma(W_i.\,[h_{t-1}, x_t] + \ b_i \quad (42)$$

For dropping the information

$$\tilde{C}_t = \tan h\,(W_C.\,[h_{t-1}, x_t] + b_c) \quad (43)$$

The output is

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (44)$$

$$o_t = \sigma|(W_o[h_{t-1}, x_t] + \ b_o) \quad (45)$$

$$h_t = o_t * \tan h\,(C_t) \quad (46)$$

The sign $*$ indicates point-wise multiplication, and the activation functions σ and tanh are used throughout. where the input, forget, and output gates are denoted by the letters i_t, f_t, and o_t, respectively, and the number of hidden layers in the cells as h_t. C_t represents the hidden layer's internal memory cell, while W_f, W_o, and W_C stand for the weighted neural network. Network traffic data is then represented by x_t, while the neural network's bias is shown by b_f and b_o.

1. Self-Attention Layer

Following the LSTM layer, introduce an attention mechanism that gives various timesteps weights according on how important they are for fault prediction. The self-attention mechanism calculates attention scores α_t for each timestep given an input sequence $\{x_1, x_2, \ldots, x_t\}$.

$$\alpha_t = SoftMax\left(W_q h_t + b_q\right) \quad (47)$$

Where W_q is a learned weight matrix, and h_t is the hidden state from the LSTM. The final output is then computed as a weighted sum of all timesteps:

$$Attention\ Output_t = \sum_{t=1}^{T} \alpha_t h_t \quad (48)$$

When a fault is detected, tasks are reallocated to available resources so that the disruption is kept at minimum. The system dynamically activates idle VMs and redistributes workloads using the HBKLO optimization algorithm while maintaining service continuity without degradation of system performance.

4. RESULT AND DISCUSSION

This paper presents Reactive Fault Tolerance in Cloud Computing Using Deep Learning and Hybrid Optimization Techniques. The performance effectiveness of the recommended technique is assessed using a cost, makespan, latency, computation time, waiting time, and energy consumption. To assess how much the newly built framework's performance has improved, it is compared to existing models, such as ABC, TLBO, PSO, BKA, and HBLKO.

a. Performance Evaluation

Table 2 compares five optimization strategies (ABC, TLBO, PSO, BKA, and HBKLO) for 100 activities based on makespan, latency, energy usage, cost, calculation time, and waiting time.

Table 2: Comparative analysis for the number of tasks 100

Optimization Method	No. of Tasks	Makes pan	Laten cy	Energy Consumption	Cost	Computation Time	Waiting Time
ABC	100	401.62	13.429 34	240.1332	484.91	0.846446	3.820546
TLBO	100	399.787 8	12.610 05	246.1231	443.05 18	0.822921	7.692522
PSO	100	305.106 4	8.4472 74	297.7044	411.37 51	0.824356	3.255369
BKA	100	326.256 7	5.4511 39	212.1525	421.57 74	0.834848	5.060929
HBKLO	100	180.734 4	2.4339 98	150.7214	295.61 91	0.800737	2.149023

ABC performs the worst, displaying inefficiency, with the highest makespan (401.62), latency, and cost. Significant delay and energy usage are displayed by TLBO, however, makespan (399.79) and cost are marginally improved. Despite using the most energy, PSO drastically lowers latency and makespan (305.11). With a moderate makespan (326.26) and energy consumption, BKA offers a decent balance, even if it has a higher latency than HBKLO. HBKLO outperforms all other methods, showing superior efficiency in all parameters, with the lowest makespan (180.73), latency, energy use, and cost.

Table 3: Comparative analysis for the number of tasks 200

Optimization Method	No. of Tasks	Makes pan	Laten cy	Energy Consumption	Cost	Computation Time	Waiting Time
ABC	200	464.587 9	15.302 31	430.1736	797.78 31	0.852468	7.832695
TLBO	200	484.388 4	14.352 48	434.2492	874.02 06	0.809833	9.821518

PSO	200	452.9316	10.37582	487.6286	860.832	0.830986	6.564737
BKA	200	472.1377	9.333709	407.6706	915.2248	0.831646	10.88642
HBKLO	200	391.0338	4.275677	319.921	710.9685	0.802866	6.047025

Table 3 compares the effectiveness of five optimization methods (ABC, TLBO, PSO, BKA, and HBKLO) for 200 tasks. Due to its high costs, latency, and makespan (464.59), ABC is inefficient. TLBO has a slightly better latency than ABC, but it has the biggest makespan (484.39) and cost. PSO has one of the highest prices and energy consumption, but it reduces latency and makespan (452.93). BKA has a greater waiting time but a reasonable makespan (472.14) and energy efficiency. With the lowest makespan (391.03), latency, energy consumption, cost, and waiting time, HBKLO is the most efficient, proving its supremacy overall.

Table 4: Comparative analysis for the number of tasks 300

Optimization Method	No. of Tasks	Makespan	Latency	Energy Consumption	Cost	Computation Time	Waiting Time
ABC	300	698.5306	17.16901	635.2095	1003.274	0.854523	12.64294
TLBO	300	708.9094	14.30482	717.9431	1092.521	0.836057	16.80088
PSO	300	609.6364	13.2127	633.9465	1028.302	0.842755	11.56242
BKA	300	655.756	12.26778	589.7643	1004.244	0.861079	13.86473
HBKLO	300	584.9157	6.116563	498.8822	931.3859	0.820726	9.655849

Table 4 compares five optimization methods (ABC, TLBO, PSO, BKA, and HBKLO) for 300 jobs. ABC and TLBO perform poorly when it comes to latency, cost, waiting time, and huge makespan (698.53 and 708.91, respectively). Although PSO has increased its makespan (609.64), it is still costly and energy-intensive. BKA balances latency and energy consumption (589.76) but lags in makespan and waiting time (655.76). With the lowest makespan (584.92), latency, energy consumption, cost, and waiting time, HBKLO surpasses all other approaches and proves its effectiveness for heavier job loads.

Table 5: Comparative analysis for the number of tasks 400

Optimization Method	No. of Tasks	Makespan	Latency	Energy Consumption	Cost	Computation Time	Waiting Time
ABC	400	831.6898	19.27315	835.9094	1307.619	0.873968	18.77259
TLBO	400	885.5101	15.23687	917.2734	1403.254	0.862816	24.10039
PSO	400	903.3117	13.15173	938.8974	1309.848	0.883519	17.05277
BKA	400	795.2192	11.06714	765.5077	1406.453	0.875925	21.96582

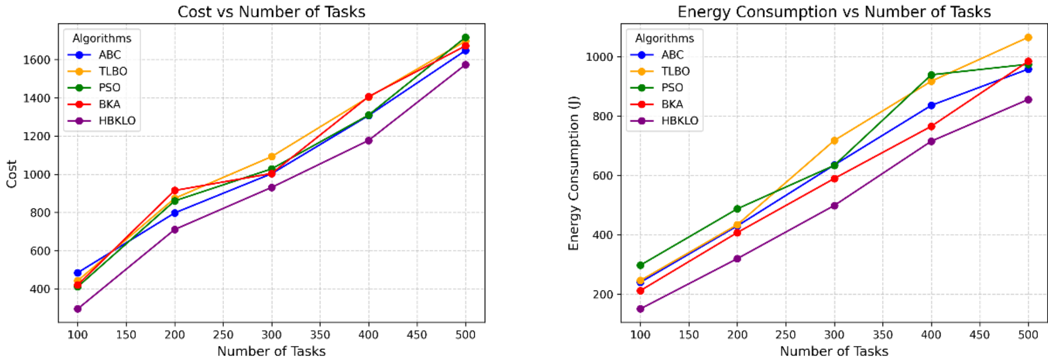
HBKLO	400	776.1887	8.002768	715.309	1177.235	0.852108	16.31117

Table 5 evaluates five optimization approaches (ABC, TLBO, PSO, BKA, and HBKLO) for 400 tasks. ABC and TLBO exhibit the most inefficient makespan (831.69 and 885.51), energy usage, cost, and waiting time. PSO consumes the most energy (938.90) but has a little shorter makespan (903.31) than TLBO. Despite having a longer makespan (795.22) and a greater energy efficiency (765.51), BKA is still quite expensive and time-consuming. HBKLO outperforms all other methods with the lowest makespan (776.19), latency, energy consumption, cost, and waiting time, proving its efficacy in handling more demanding workloads.

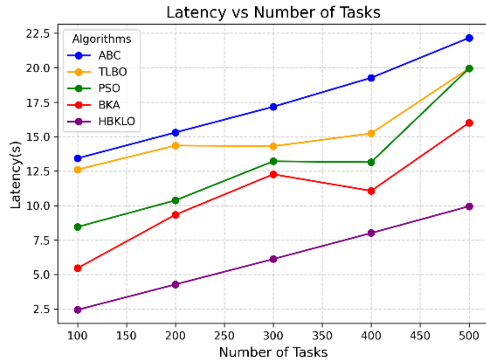
Table 6: Comparative analysis for the number of tasks 500

Optimization Method	**No. of Tasks**	**Makespan**	**Latency**	**Energy Consumption**	**Cost**	**Computation Time**	**Waiting Time**
ABC	500	1065.141	22.17053	958.1304	1647.933	0.906379	25.84803
TLBO	500	1111.124	19.96067	1065.218	1697.217	0.893284	26.83158
PSO	500	1142.187	19.95881	974.3827	1717.07	0.892397	24.61508
BKA	500	1064.068	16.00869	984.3103	1672.368	0.883497	28.69924
HBKLO	500	959.9524	9.948053	855.7917	1574.114	0.856079	22.27192

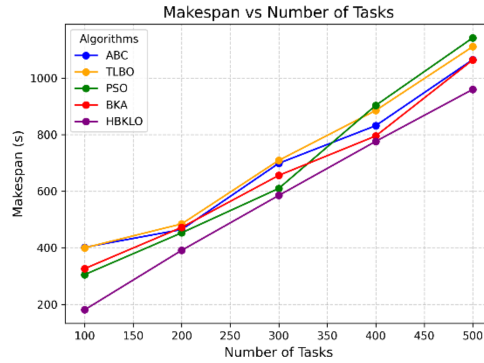
Table 6 compares five optimization methods (ABC, TLBO, PSO, BKA, and HBKLO) for 500 tasks. ABC, TLBO, and PSO display high makespan (1065.14, 1111.12, and 1142.19, respectively), latency, energy usage, and cost, with PSO displaying the highest cost (1717.07). BKA has a high waiting time (28.70), but its makespan (1064.07) and energy consumption are slightly lower. The most effective option for managing large-scale jobs is HBKLO, which has the lowest makespan (959.95), latency, energy usage, cost, and waiting time.



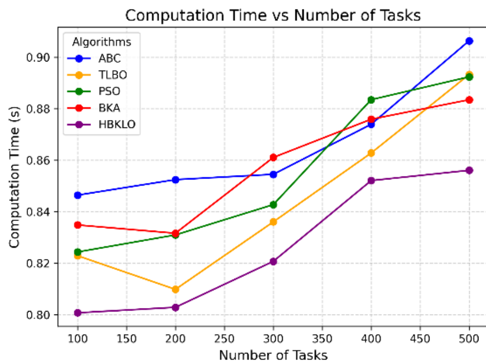
a. Cost **b.** Energy Consumption



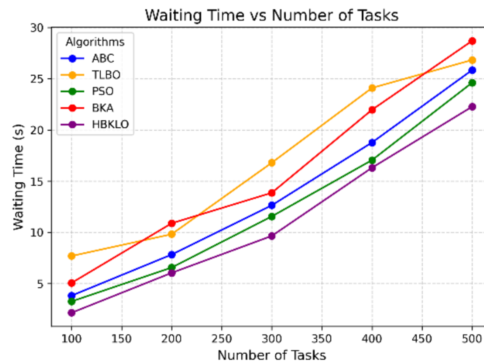
c. Latency



d. Makespan



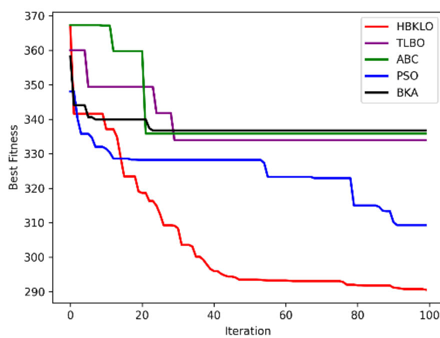
e. Computation time

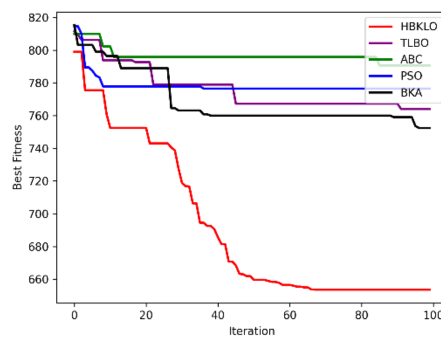


f. Waiting time

Figure 2 (a)-(f): Graphical representation of the several performances with proposed and existing works

The performance metrics, which include cost, energy consumption, latency, computation time, waiting time, and Makespan are represented graphically in Figures 2 (a) to (f). These metrics are compared with several approaches, including ABC, TLBO, PSO, BKA, and HBLKO.





Vol. 22, No. 1, (2025)
ISSN: 1005-0930

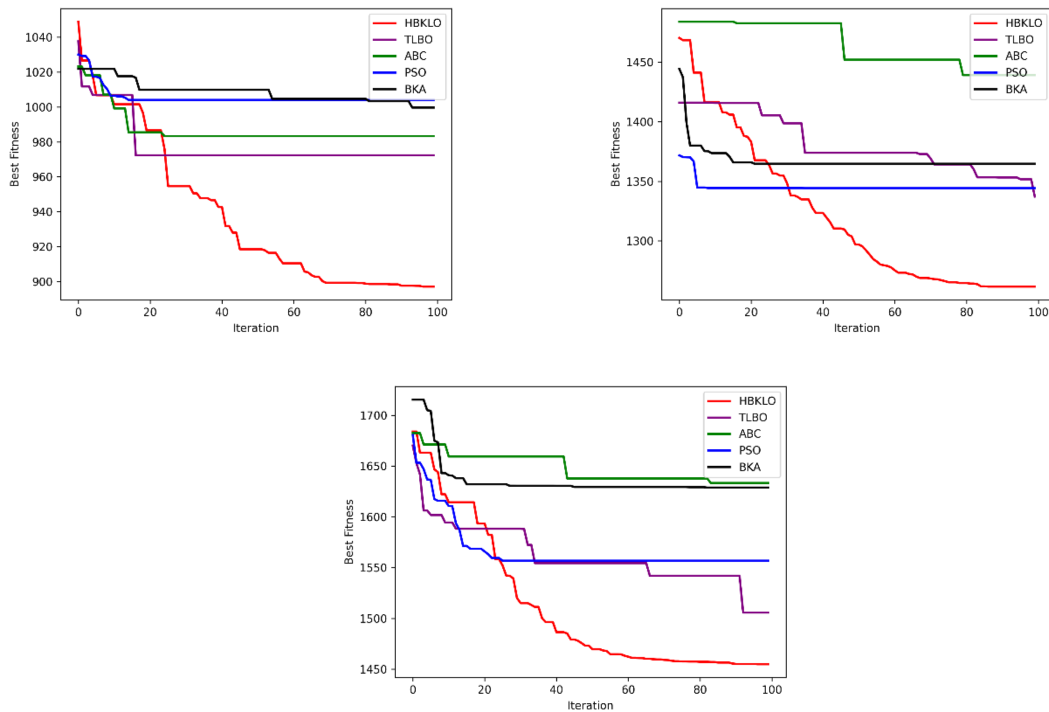


Figure 3: Graphical representation of the convergence graphs

Figure 3 demonstrates the Graphical illustration of the convergence curves for various fitness sizes, comparing HBKLO with existing works like ABC, TLBO, PSO, and BKA. Fig 4 (a- e) represents the Graphical representation of the Scheduling graph with various nodes.

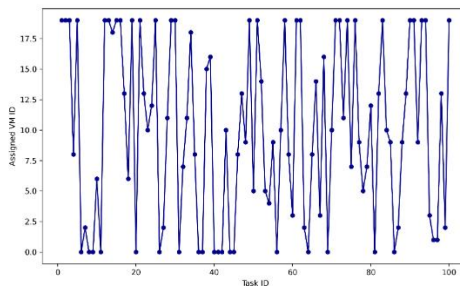


Figure 4 (a): Graphical representation of the Scheduling graph with 100 nodes

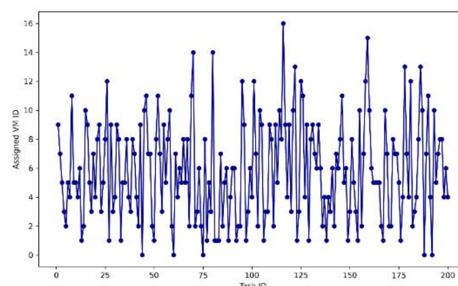


Figure 4 (b): Graphical representation of the Scheduling graph with 200 nodes

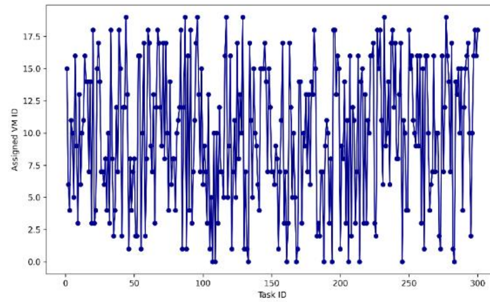


Figure 4 (c): Graphical representation of the Scheduling graph with 300 nodes

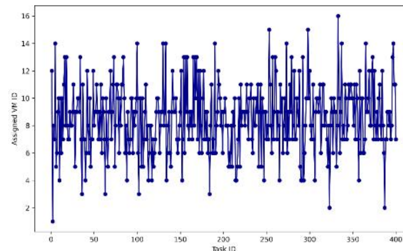


Figure 4 (d): Graphical representation of the Scheduling graph with 400 nodes

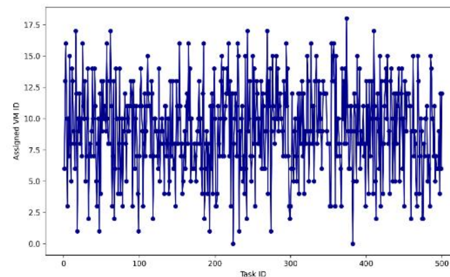


Figure 4 (e): Graphical representation of the Scheduling graph with 500 nodes

b. Fault tolerance related metrics

The table evaluates the performance of various optimization methods—ABC, TLBO, PSO, BKA, and HBKLO—based on three key metrics: Failure Ratio (FR), Failure Slowdown (FD), and Performance Improvement Rate (PIR). These metrics provide insights into the reliability, efficiency, and overall effectiveness of each method.

i. Failure Ratio (FR)

The FR is the ratio of failures that have occurred in the execution of different optimization methods. It can be used to indicate how reliable each method is, since lower values of FR point out fewer failures. In the given table, the FR ranges from 0.05295 to 0.33464, which points out variation in performance across the methods. Among the optimization techniques included, HBKLO steadily holds the lowest FR throughout the runs, which in all cases is very close to zero (e.g., 0.05295, 0.11457, 0.17137, and 0.28125). Other techniques, such as ABC, TLBO, PSO, and BKA, have higher FR values with increased chances of failure when task complexity increases, illustrated in Figure 5.

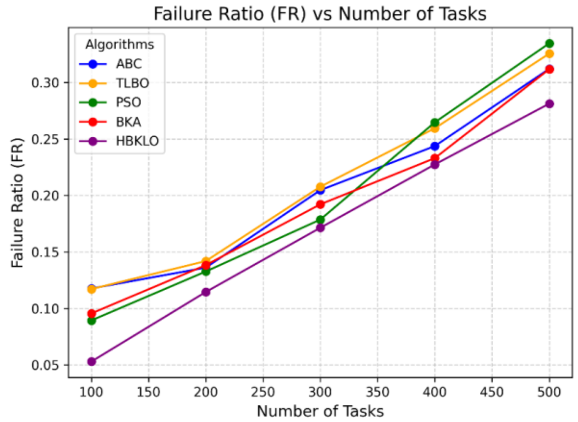


Figure 5: Comparison of FR rate

ii. Failure Slowdown (FD)

The FD refers to the extent to which failures retard the performance or efficiency of an optimization method. Large FD values imply that delays or inefficiencies due to failures are high, but lower values suggest that methods are effective in handling failure at the cost of minimal reductions in performance. From the table, it can be seen that FD varies from 0 to 1. There are methods such as HBKLO that often resulted in lower values such as 0, 0.5005, 0.6674, and 0.8032 that imply better resistance to failure. Methods like ABC and TLBO had a tendency of higher FD values such as 0.8850, 0.9443, and 1 which imply a higher performance slowdown due to failures. This indicates that HBKLO is more efficient in reducing the impact of failures compared to the other optimization techniques as illustrated by Figure 6.

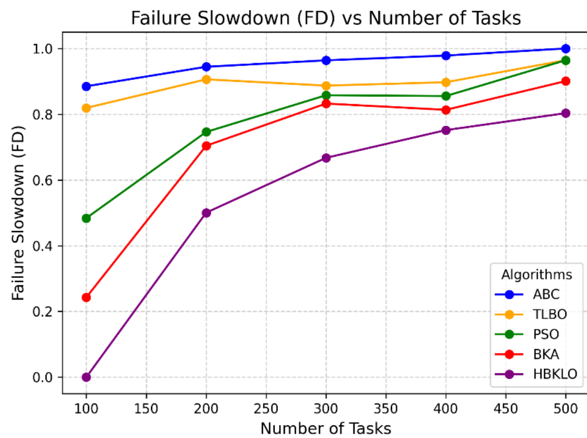


Figure 6: Comparison of FD rate

iii. Performance Improvement Rate (PIR)

The performance improvement rate shows how effective the optimization method is for enhancing performance; a higher value reflects a better improvement. PIR=1 signifies the maximum performance improvement; values below that indicate that efficiency or success in obtaining the optimal result is declining. In the table, the values of PIR are varied between 0 and 1. HBKLO mainly obtains higher values like that of 1, 0.7813, 0.5796, and 0.3807. This implies its capability to enhance the performance was better. Other techniques like ABC and TLBO mostly represent lower values of PIRs, such as 0.7048, 0.6842, and 0.0323, showing failures and inefficiency increase in the process. This means HBKLO does not only alleviate failures well but

also performs better and more consistently as compared to other optimization methods, which is depicted in Figure 7.

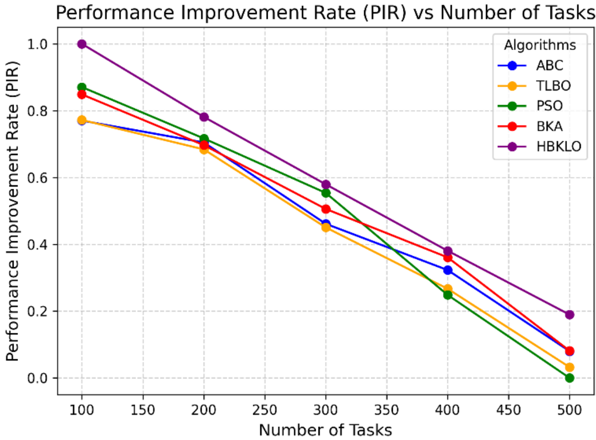


Figure 7: Comparison of PIR values

iv. Resource Utilization

The "Resource Utilization" column lists numerical values that represent how much resource (likely computational or memory) is consumed by different optimization techniques. The optimization methods include Artificial Bee Colony (ABC), Teaching-Learning-Based Optimization (TLBO), Particle Swarm Optimization (PSO), Bacterial Foraging Algorithm (BKA), and a hybrid version of BKA, termed Hybrid BKLO. The resource utilization varies across these methods, with ABC and HBKLO showing higher resource usage in most instances, while TLBO and PSO tend to consume fewer resources. For example, ABC and HBKLO methods have higher values such as 5.196 and 13.290, respectively, indicating their more intensive resource consumption, whereas methods like PSO and BKA exhibit lower values like 3.139 and 2.968, reflecting a more efficient use of resources. This highlights the trade-offs between optimization method effectiveness and resource demands. Fig 8 shows the comparison of Resource Utilization values.

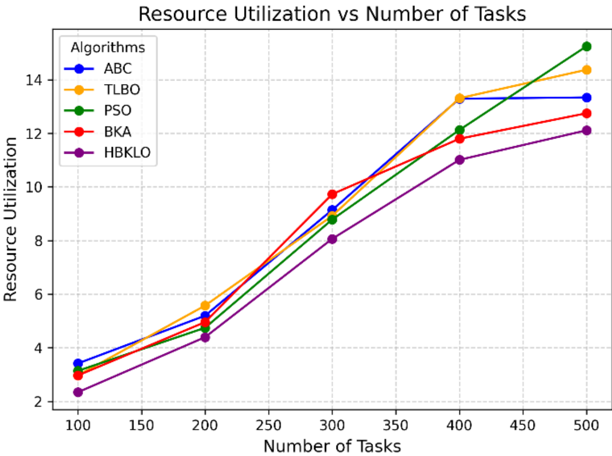


Figure 8: Comparison of Resource Utilization values

c. Overall accuracy and loss

Figure 9 demonstrates the performance for different numbers of nodes 100, 200, 300, 400, and 500 are depicted graphically in the scheduling graphs. The scalability and efficacy of the scheduling techniques are

Vol. 22, No. 1, (2025)
ISSN: 1005-0930

demonstrated by these graphs, which offer a comparative study of assigned tasks and efficiency as the number of nodes rises.

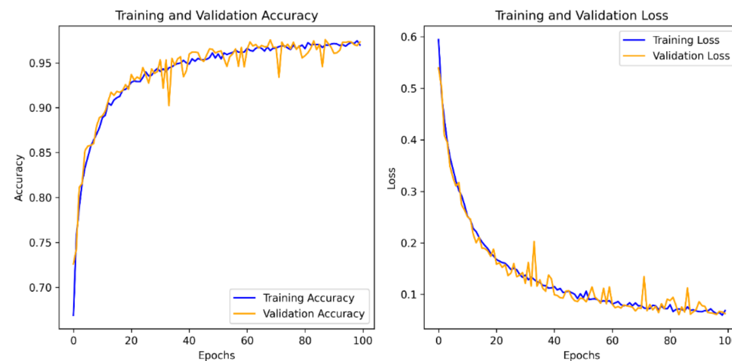


Figure 9: Training and validation graph for Epoch's accuracy and loss

5. CONCLUSION

This paper discusses a comprehensive framework of reactive fault tolerance in cloud computing by incorporating deep learning and hybrid optimization techniques for improving the reliability, efficiency, and scalability of the system. It involves the use of an M-LSTM model for proactive prediction of faults so that failure occurrences are timely detected. The HBKLO algorithm is used to optimize tasks in scheduling and failure recovery as well, which balances those critical objectives of cost, makespan, and energy consumption. The DCRL load balancer ensures an optimal utilization of resources. It dynamically reallocates the tasks across virtual machines on real-time. Further, the fault-tolerant mechanism provides system resilience in the form of recovery from failure, which happens very rapidly with minimal interference. Through experimental results, it has been shown that the proposed framework enhances the performance of the system, reduces energy usage, and minimizes operational costs in dynamic cloud environments. This integrated approach provides a scalable, cost-effective, and energy-efficient solution for handling the complexity of modern cloud computing systems.

REFERENCE

[1] Z. Ahmad, A.I. Jehangiri, N. Mohamed, M. Othman, and A.I. Umar, Fault tolerant and data oriented scientific workflows management and scheduling system in cloud computing. IEEE Access, 10, (2022) pp.77614-77632.

[2] M.I. Alghamdi, Optimization of load balancing and task scheduling in cloud computing environments using artificial neural networks-based binary particle swarm optimization (BPSO). Sustainability, 14(19), (2022) p.11982.

[3] K. Devi, and D. Paulraj, Multilevel fault-tolerance aware scheduling technique in cloud environment. Journal of Internet Technology, 22(1), (2021) pp.109-119.

[4] M.K. Dixit, and D. Kumar, Fault tolerant aware energy-efficient VM integration in Cloud Computing (2023).

[5] N. Hagshenas, M. Mojarad, and H. Arfaeinia, A fuzzy approach to fault tolerant in cloud using the checkpoint migration technique. International journal of intelligent systems and applications, 12(3), (2022) p.18.

[6] U.K. Jena, P.K. Das, and M.R. Kabat, Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. Journal of King Saud University-Computer and Information Sciences, 34(6), (2022) pp.2332-2342.

[7] T. Jin, and B. Zhang, Intermediate data fault-tolerant method of cloud computing accounting service platform supporting cost-benefit analysis. Journal of Cloud Computing, 12(1), (2023) p.2.
[8] C. Kalaskar, and S. Thangam, Fault tolerance of cloud infrastructure with machine learning. Cybernetics and Information Technologies, 23(4), (2023) pp.26-50.
[9] B. Kruekaew, and W. Kimpan, Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning. IEEE Access, 10, (2022) pp.17803-17818
[10] M. Kumar HS, S.S. Mustapha, P. Gupta, and R.P. Tripathi, Intelligent Fault-Tolerant Mechanism for Data Centers of Cloud Infrastructure. Mathematical Problems in Engineering, 2022(1), (2022) p.2379643.
[11] A. Mehta, D. Goyal, A. Choudhary, B.S. Pabla, and S. Belghith, Machine Learning-Based Fault Diagnosis of Self-Aligning Bearings for Rotating Machinery Using Infrared Thermography. Mathematical Problems in Engineering, 2021(1), (2021) p.9947300.
[12] S.U. Mushtaq, S. Sheikh, and S.M. Idrees, Next-Gen Cloud Efficiency: Fault-Tolerant Task Scheduling with Neighboring Reservations for Improved Cloud Resource Utilization. IEEE Access (2024).
[13] P. Nama, M. Bhoyar, S. Chinta, and P. Reddy, Optimizing Database Replication Strategies through Machine Learning for Enhanced Fault Tolerance in Cloud-Based Environments. Machine learning (ML), 63(3) (2023).
[14] A. Rawat, R. Sushil, A. Agarwal, A. Sikander, and R.S. Bhadoria, A new adaptive fault tolerant framework in the cloud. IETE Journal of Research, 69(5), (2023) pp.2897-2909.
[15] A.U. Rehman, R.L. Aguiar, and J.P. Barraca, Fault-tolerance in the scope of cloud computing. IEEE Access, 10, (2022) pp.63422-63441.
[16] R. Rengaraj alias Muralidharan, and K. Latha, Gorilla Troops Optimizer Based Fault Tolerant Aware Scheduling Scheme for Cloud Environment. Intelligent Automation & Soft Computing, (2023) 35(2).
[17] A. Rezaeipanah, M. Mojarad, and A. Fakhari, Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic. International Journal of Computers and Applications, 44(2), (2022) pp.139-147.
[18] V. Sathiyamoorthi, P. Keerthika, P. Suresh, Z.J. Zhang, A.P. Rao, and K. Logeswaran, Adaptive fault tolerant resource allocation scheme for cloud computing environments. Journal of Organizational and End User Computing (JOEUC), 33(5), (2021) pp.135-152.
[19] D. Saxena, I. Gupta, A.K. Singh, and C.N. Lee, A fault tolerant elastic resource management framework toward high availability of cloud services. IEEE Transactions on Network and Service Management, 19(3), (2022) pp.3048-3061.
[20] A. Semmoud, M. Hakem, B. Benmammar, and J.C. Charr, A new fault-tolerant algorithm based on replication and preemptive migration in cloud computing. International Journal of Cloud Applications and Computing (IJCAC), 12(1), (2022) pp.1-14.
[21] S. Setaouti, D.A. Bensaber, R. Adjoudj, and M. Rebbah, Fault tolerance directed by service level agreement in cloud computing environments. Int. J. Com. Dig. Sys, (2022) 11(1).
[22] M. Shaukat, W. Alasmary, E. Alanazi, J. Shuja, S.A. Madani, and C.H. Hsu, Balanced energy-aware and fault-tolerant data center scheduling. Sensors, 22(4), (2022) p.1482.
[23] M. Shiva Rama Krishna, and S. Mangalampalli, A Novel Fault-Tolerant Aware Task Scheduler Using Deep Reinforcement Learning in Cloud Computing. Applied Sciences, 13(21), (2023) p.12015.
[24] M. Uppal, D. Gupta, S. Juneja, A. Sulaiman, K. Rajab, A. Rajab, M.A. Elmagzoub, and A. Shaikh, Cloud-based fault prediction for real-time monitoring of sensor data in hospital environment using machine learning. Sustainability, 14(18), (2022) p.11667.

[25] J. Zhao, Y. Ding, Y. Zhai, Y. Jiang, Y. Zhai, and M. Hu, Explore unlabeled big data learning to online failure prediction in safety-aware cloud environment. Journal of Parallel and Distributed Computing, 153, (2021) pp.53-63.